센서 네트워크를 위한 동적 재구성 가능한 유한 상태 기반 운영체제 구현 및 평가

김도혁○, 김태형
한양대학교 컴퓨터공학과
{dohyuk○, tkim}@cse.hanyang.ac.kr

Implementation and Evaluation of the Finite State-driven Operating System for Dynamically Reconfigurable Sensor Networks

Do-Hyuk Kim○, Tae-Hyung Kim
Dept. of Computer Science & Engineering, Hanyang University

요 약

저 전력 마이크로 프로세서와 무선 통신 모듈, 통합된 시스템 장치들을 내장한 센서 노드를 이용하여 구축된 센서 네트워크는 동작환경과 구조가 특이하며 시스템 설계 시 많은 제약 조건들이 고려 되어야 한다. 이러한 센서 네트워크의 각 노드는 센서를 이용한 정보 수집과 같은 여러 가지 응용 프로그램, 노드 간 통신, 제한된 시스템 자원을 효율적으로 관리 할 수 있는 기능들이 필요하며 이에 센서 노드를 위한 여러 가지 운영체제들이 제안되었다. 본 논문에서는 센서 노드를 위해 효과적이고 응용의 변화에 대처할 수 있도록 동적 재구성 기능을 지원하도록 설계된 유한 상태 머신 (finite state machine) 기반의 운영체제인 SenOS의 특징과 구현된 SenOS를 개선한 결과를 발표한다.

1. 서 론

무선 센서 네트워크는 기존의 컴퓨팅 시스템과는 달리 극도로 제한된 시스템 자원과 열악한 환경 속에서 동작하며 저 전력 마이크로 프로세서와 무선통신 모듈, 센서 등을 통합한 시스템이 내장된 센서 노드를 이용하여 구성하게 된다 [1].

센서 네트워크에서 센서 노드는 정보 수집, 데이터 통신, 협력을 통한 모니터링과 같은 여러 작업들을 수행하게 된다. 이러한 작업들을 자원이 제한적인 센서 노드의 하드웨어에서 효과적으로 수행 하기 위해 센서 노드를 위한 운영체제가 필요하게 되었고 센서 노드에 적합한 태스크 구조 및 관리 기능, 센서 노드간의 효율적인 에너지 관리, 센서 네트워크 응용에 적합한 메시지 처리 기능, 저전력 관리 기능, 환경 변화에 따른 재구성, 센서 응용 개발환경 등에 초점을 두고 연구가 진행중인 운영체제들이 있다.

본 논문에서는 센서 네트워크 노드에 적합한 운영체제를 디자인하기 위해 유한 상태 머신 기반의 실행모델을 이용하고 응용의 변화에 대처할 수 있게 동적 재구성 기능을 지원하도록 설계된 SenOS의 구조에 대해 기술하고 코드크기, 전력소모, 반응성과 같은 다양한 요구 조건들이 어떻게 SenOS 구현에 반영되었는지 살펴 본다.

2. 관련연구

센서 네트워크 노드를 위한 운영체제는 이벤트 구동 방식을 사용하는 운영체제와 멀티 태스킹과 멀티 쓰레드를 사용하는 운영체제로 나누어 볼 수 있다. 이 두 방식의 두드러진 차이점으로 스케쥴링 방식으로 이벤트 구동 방식의 경우 high-low, task-event와 같이 2 레벨의 우선순위를 갖는 FIFO 스케쥴링 기법을 사용하고, 멀티 태스킹과 멀티 쓰레드 방식을 사용하는 운영체제에서는 프로세서의 타임 슬라이스를 이용하기 위해 라운드 로빈 방식의 스케쥴링이나 EDF와 같은 스케쥴링을 사용한다. 후자의 경우 현재 실행 중인 쓰레드의 상태를 스택에 저장하고 스위칭을 하기 때문에 스택 포인터, 스택 범위 정보, 쓰레드가 시작되는 함수의 포인터 등을 유지하는 것이 필요하고 이에 따른 추가적인 메모리와 프로세서 사용으로 더 큰 오버헤드를 갖게 된다.

이벤트 구동 방식의 대표적인 예로 TinyOS가 있으며 상태 머신 모델링을 이용하여 설계 되었다 [2]. TinyOS는 센서 노드에 최적화 시킨 아주 작은 운영체제이지만 실행 도중에 동적으로 센서 노드를 재구성하기가 어렵다. TinyOS는 C를 확장한 nesC를 이용하여 구현되었으며 컴포넌트 및 애플리케이션을 개발할 때 nesC를 이용하여야 하는데 이는 nesC를 처음 접하는 개발자들에게 새롭게 언어를 습득하여야 하는 부담감을 줄 수 있다.

이벤트 구동 방식으로 동작하는 또 다른 운영체제에는 유한 상태 머신 기반의 실행 모델을 이용하여 설계한 SenOS가 있다 [4]. SenOS는 C로 구현되었고 응용 프로그램은 센서 노드의 태스크를 상태 모델링을 통해 상태 전이 테이블로 구성한다 [5]. 교체 가능한 상태 전이 테이블은 실행 중에 동적으로 재구성될 수 있으며 개발자는 상태 머신 기반 시스템 모델을 디자인 할 수 있는 툴을 이용하여 보다 쉽게 응용 프로그램을 개발 할 수 있다.

멀티 쓰레드를 지원하는 MantisOS는 응용개발 프로세스의 단순화를 통하여 센서 네트워크를 효율적으로 구축하고자 하였고 개발자에게 친숙한 윈도우나 리눅스 개발 환경에서의 C API와 같은 이식성 있는 프로그래밍 언어의 특성을 이용하여 설계되었다 [3]. 프로그램 개발 시에 크로스 플랫폼을 지원하여 같은 코

드가 다양한 플랫폼 상에서 실행 될 수 있도록 하였지만 프로그램을 실행 중에 동적으로 재구성 할 수는 없다.

3. 상태 기반 운영체제 구현

본 연구에서는 센서 네트워크 운영체제로 제안된 SenOS의 설계 개념을 이용하고 기존에 구현된 구조에서 시스템 운영에 필요한 부분들을 추가하고 수정해서 성능을 향상시켰다. SenOS 구현과 실험에는 8-bit MCU ATmega128L과 실내에서 50m에 무선 송신 범위를 갖는 CC2420 (Zigbee) RF 트랜시버를 장착한 센서 노드를 이용하였다. 이 센서 노드는 내부적으로 128KByte의 프로그램 메모리, 2KByte SRAM, 4KByte EEPROM을 갖는다. 추가적으로 센서 데이터를 장시간 저장하는데 사용 할 수 있는 512KByte의 Flash 메모리가 외부에 부착되어 있으며 온도, 습도, 조도, 적외선을 측정 할 수 있는 센서를 갖고 있다.

3.1 개선된 SenOS 구조

이벤트를 처리하는 상태 정렬기와 이벤트 큐로 이루어진 커널, 상태 전이 테이블, 출력 함수들을 담고 있는 콜백 라이브러리는 기존에 발표된 SenOS와 같다. 위 개념을 효과적으로 구현하기 위해 이벤트 생성시 메모리를 할당해 주는 이벤트 풀, 동적 재구성시 새로운 상태 테이블을 저장하는데 필요한 메모리를 할당해 주는 테이블 풀과 무선이나 시리얼 케이블을 이용하여 전달 된 데이터를 처리하는 메세지 파서, 인터럽트가 발생되면 호출되어 해당 인터럽트를 처리하고 이벤트를 생성하는 인터럽트 핸들러, 동적으로 테이블을 관리하기 위해 사용되는 테이블 로더가 추가적으로 구현되었다. 이러한 개선 작업은 모듈화 과정을 통해 응용에 따라 선택적으로 시스템을 구성 할 수 있도록 refactoring 하였다. 그림1은 개선된 SenOS의 구조이다.

3.2 모듈화된 구조를 이용한 시스템 구성

센서 네트워크 환경에 적합한 운영체제를 구현하기 위해서는 제한된 메모리에 적합한 작은 크기의 운영체제가 되도록 해야 한다. 이를 위해 센서 응용에 따라 시스템을 재구성 가능한 형태가 되도록 하였고 senos_cfg.h 파일 내에 전 처리기를 이용하여 필요한 모듈들을 활성화 및 비활성화 시킬 수 있다. 모듈화된 구조를 이용하여 응용 개발시 필요한 모듈만으로 시스템을 구성할 수 있고 이를 통해 시스템 자원을 절약 할 수 있게 된다.

3.3 이벤트 큐와 스케쥴링

이벤트 구동 방식의 SenOS를 위해 동기적 혹은 비동기적으로 발생하는 이벤트를 저장 할 이벤트 큐가 필요하다. 이벤트 큐는 이중 연결 리스트를 이용하여 이벤트를 관리하고, 사용된 이벤트의 갯수, 큐에 남아 있는 이벤트의 갯수 정보를 가지며 이러한 항목들은 시스템 디버깅 정보로 이용 될 수 있다. 이벤트 큐는 이전 시스템과 달리 2단계의 우선 순위를 갖는 FIFO 방식으로 개선하였다. 일반적인 이벤트들은 FIFO 방식으로 저장하며 긴급하게 처리되어야 할 이벤트는 이벤트 생성시 LIFO 방식으로 큐에 삽입 할 수 있도록 함으로써 긴급한 센서 데이터 처리를 위한 프로그래밍이 가능하도록 하였다.

3.4 이벤트와 이벤트 풀

이벤트는 상태 머신의 상태를 결정하는 주체로써 시스템 내에서 가장 빈번하게 처리된다. 이러한 이벤트들은 시스템이 구동 중일 때 얼마나 발생 할지 예측하기 어려우므로 이벤트 생성 요청 시에 동적으로 메모리를 할당 할 필요가 있다.

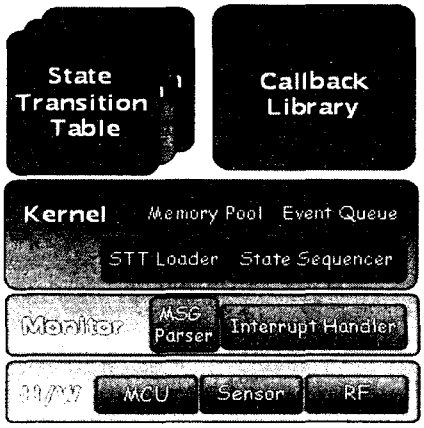

그림 1. 개선된 SenOS의 구조

일반적으로 센서 노드들은 시스템 리셋 없이 오랜 기간 동안 안정적으로 실행되어야 하는데 RAM을 사용한 힙 기반 메모리 할당은 단편화, 메모리 누수, 복잡한 힙 검사 알고리즘으로 인한 실행 시간의 오버헤드, 댕글링 포인터와 같은 문제점을 야기시켜 시스템의 안정성을 낮추게 된다. 이를 해결하기 위해 시스템 초기화 때 사용자가 시스템 운영에 필요한 메모리를 설정하도록 하고 정적 메모리 공간을 사용하였다. 이벤트 풀은 이렇게 사용자가 지정한 연속된 메모리 공간을 이벤트의 크기만큼 나누어 미리 메모리 블록을 만들어 놓고 이벤트 생성시 이를 할당 해주며 사용이 완료된 이벤트는 사용한 메모리 블록을 반환하게 된다.

3.5 상태 전이 테이블과 동적 재구성을 위한 테이블 풀

상태 전이 테이블은 센서 노드의 기능을 결정하는 응용 프로그램이다. 이러한 상태 전이 테이블은 교체 가능하게 되어야 하며 동적 재구성 시 이를 적절히 관리 할 수 있는 모듈이 필요하다. 이를 위해 추가한 방법으로 시스템 구성 설정을 통해 동적 재구성 기능을 활성화 시키면 응용 프로그램에 필요한 메모리를 할당해 주기 위한 테이블 풀이 만들어지고, 메모리 할당과 해제를 수행하는 함수들이 활성화되도록 하였다. 시스템이 동작 중에 응용 업데이트나 응용 프로그램 추가 메시지를 수신할 경우 테이블 풀에서 요청한 크기만큼 메모리를 할당해 주고 무선 수신된 데이터에서 메세지 파서를 통해 응용 프로그램 데이터를 추출하여 테이블에 각 항목을 설정한 후 상태 전이 테이블 로더가 시스템에 새로운 테이블을 등록 시키는 방법으로 구현하였다.

3.6 유저 콜백과 시스템 콜백 함수

콜백 함수는 센서 노드의 기능을 결정할 때 사용하는데 기존 시스템과 달리 시스템에서 제공되는 시스템 콜백 함수 외에도 사용자가 시스템 API를 이용하여 콜백 함수를 작성 할 수 있도록 한 것이다. 사용할 콜백 함수는 시스템 초기화 시 등록된다.

3.7 인터럽트 핸들러

인터럽트 핸들러는 하드웨어에서 발생한 인터럽트들을 관리하

고 해당 인터럽트를 이벤트로 변경한다. 이전 구현에서는 인터럽트 필터를 통해 이를 처리하였지만 현재 구현된 시스템에서는 모듈화를 유지하고 일관된 인터페이스를 제공하기 위해 인터럽트 자체는 디바이스 드라이버 내에서 처리하되 각 드라이버 마다 외부로 전달되는 인터럽트 핸들러를 두어 이를 처리하게 하였다. 예를 들어 하드웨어 타이머 인터럽트가 발생되면 드라이버 외부로 전달되는 인터페이스를 통해 모니터 내에 구현되어 있는 핸들러 함수가 호출되고 타이머 이벤트를 생성하게 된다.

3.8 파워 관리

파워 관리는 이전 구현과 동일하며 파워를 절약하기 위해서 이벤트 큐에 이벤트가 존재하지 않을 때 하드웨어에서 제공하는 기능을 이용하여 슬립 모드로 전환 할 수 있는 기능을 추가하였다.

4. SenOS를 이용한 응용 프로그램 작성

그림2는 1초 마다 조도를 센싱하는 센서 노드의 태스크를 상태 모델링을 통해 상태 전이 테이블로 구성하여 응용 프로그램을 작성한 예이다. senos_cfg.h 파일에 사용 할 모듈을 정의하였고 3개의 시스템 콜백 함수를 사용하였다. 위의 응용 프로그램은 13KByte정도에 코드 크기를 갖는다.

5. TinyOS와의 비교

SenOS 운영체제의 전반적인 성능 평가는 현재 연구가 진행중에 있지만, 본 논문에서는 그림2에 나타난 응용프로그램을 작동하기 위해 필요한 기본적인 크기 정보를 획득하여 표1과 같이 정리하였다. 동일한 응용 프로그램과 하드웨어 플랫폼을 이용하였으며 단지 운영체제만 바뀐 상태이다.

구분	TinyOS	SenOS
CODE SIZE	4000 byte	5830 byte
ROM	1444 byte	2152 byte
RAM	44 byte	225 byte
Dynamic Reconfigurable	NO	YES

표 1. TinyOS와 SenOS 크기 비교

운영체제 자체의 코드 크기는 SenOS가 조금 크지만 센서 노드의 메모리 보유량을 감안할 때 문제가 없는 수준이다. 메모리 요구량 역시 SenOS가 다소 크지만 이것은 센서 네트워크 응용에 필수적인 동적 재구성성을 감안하고 CASE 툴과의 연계사용을 통한 개발편의성 및 코드 안정성을 얻기 위해 필요한 최소한의 비용이며 센서 노드 HW 구성상 전혀 무리가 없는 수준임을 알 수 있다. 중요한 것은 이와 같은 이점들이 코드 최소화를 목표로 한 TinyOS와 비교해 보아도 큰 차이가 없는 수준이라는 점이다. 실제로 TinyOS에 동적 재구성성을 지원하기 위해 VM을 추가한 Mate는 이보다 훨씬 크기를 갖게 되며 동작속도상 현저한 성능저하를 보이게 된다. 향후 SenOS의 동적 재구성 기능의 효율성을 평가하기 위해 TinyOS 상에서 동적 재구성을 지원해주는 Mate와 비교해 보고 라우팅 알고리즘을 적용한 응용 프로그램을 이용하여 구현된 SenOS가 갖는 장점들을 비교 평가할 예정이다.

6. 결론

구현된 SenOS는 모듈화를 통한 시스템 구성, 파워 관리, 유한 상태 머신 컴퓨팅, 동적 재구성 기능을 제공 한다. 향후 기존에 연구 중인 운영체제의 특징들을 분석하여 장, 단점을 비교해 본 후 구현된 SenOS가 센서 네트워크 노드에 적합한 운영 체제가 될 수 있게 보완하려 한다.

```
#include "../../senos/include/senos_cfg.h"

#define EVENT_BUF_SIZE  100
#define EVENTPOOLNUM    0
char eventBuf[EVENT_BUF_SIZE];

int main(void)
{
    STT stt[STT_TABLE_SIZE]= {
        //current state, signal       , dest_state   , action
        {STATE_IDLE   , REPEAT_TIMER   , STATE_TIMER , TIMER_START}    ,
        {STATE_TIMER , START_SENSING  , STATE_PHOTO , PHOTO_SENSING}  ,
        {STATE_PHOTO , END_SENSING    , STATE_TIMER , PHOTO_GET_DATA} ,
    };

    callback uselib[CALLBACK_SIZE];

    // Set system shared data
    g_sysRes.timer.timerSignal = START_SENSING;
    g_sysRes.timer.delay = 1000;
    g_sysRes.timer.timerType = REPEAT;
    g_sysRes.timer.timerNum = TIMER0;

    // Make event pool
    eventMakePool(eventBuf, EVENT_BUF_SIZE, 4, EVENTPOOLNUM);

    // Use system callback library
    uselib[TIMER_START].Func = callbackHandleTimer;
    uselib[PHOTO_SENSING].Func = callbackHandlePhoto;
    uselib[PHOTO_GET_DATA].Func = callbackSendPhotoData;

    senosInit();    // Init system variable and H/W resource
    senosSTTLoader(STT_TABLE_SIZE, stt);
    senosInstallCallback(CALLBACK_SIZE, uselib);    // Register and callback lib
    eventCreate(REPEAT_TIMER);  // start Signal
    senosStart();
```

그림 2. SenOS를 이용한 응용 프로그램 작성 예

참고 문헌

[1] I.F.Akyildiz, W. Su et al., " A Survey on Sensor Networks" , IEEE Communications magazine, August 2002.

[2] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, " System Architecture Directions for Networked Sensors" International Conference on Architectural Support for Programming Languages and Operating Systems (2000).

[3] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, J. Deng, R. " Han MANTIS: System Support for Multimodal Networks of In-situ Sensors" Appeared in 2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA) 2003.

[4] Seongsoo Hong ; T.-H. Kim, "SenOS: state-driven operating system architecture for dynamic sensor node reconfigurability", *International Conference on Ubiquitous Computing*, pp.201-203, Oct. 2003.

[5] T.-H. Kim ; Seongsoo Hong, "State machine based operating system architecture for wireless sensor networks", *Lecture Notes in Computer Science*, vol. 3320 no. pp.803-803, Dec. 2004